A MULTI-EXCHANGE HEURISTIC FOR FORMATION OF BALANCED DISJOINT RINGS

A Thesis

by

SARATH K. SASI KUMAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2005

Major Subject: Industrial Engineering

A MULTI-EXCHANGE HEURISTIC FOR FORMATION OF BALANCED DISJOINT RINGS

A Thesis

by

SARATH K. SASI KUMAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Halit Uster
Committee Members,	Sila Cetinkaya
	Michael A. Bevan
Head of Department,	Brett A. Peters

August 2005

Major Subject: Industrial Engineering

ABSTRACT

A Multi-Exchange Heuristic for Formation of
Balanced Disjoint Rings. (August 2005)
Sarath K. Sasi Kumar, B.E., Regional Engineering College, Trichy, India
Chair of Advisory Committee: Dr. Halit Uster

Telecommunication networks form an integral part of life. Avoiding failures on these networks is always not possible. Designing network structures that survive these failures have become important in ensuring the reliability of these network structures. With the introduction of SONET (Synchronous Optical Network) technology, rings have become the preferred survivable network structure. This network configuration has a set of disjoint rings (each node being a part of single ring), and these disjoint rings are connected via another main ring. In this research, we present a mathematical model for the design of such disjoint rings with node number balance criterion among the rings. When, given a set of nodes and distances between them, the Balanced Disjoint Rings (BDR) problem is the minimum total link length clustering of nodes into a given number of disjoint rings in such a way that there is almost the same number of nodes in each ring. The BDR problem is a class of the standard Traveling Salesman Problem (TSP). It is clear from this observation that the BDR problem becomes a TSP when the number of rings required is set to one. Hence BDR is NP-Hard, and we do not expect to obtain a polynomial time algorithm for its solution. To overcome this problem, we developed a set of construction heuristics (Break-MST, Distance Method, Hybrid Method, GRASP-Based Distance Method) and improvement heuristics (Multi-Exchange, Single Move). Different combinations

of construction and improvement heuristics were implemented and the quality of solution thus obtained was compared to the standard Branch and Cut Technique. It was found that the algorithm with GRASP-Based Distance Method as the construction heuristic and multi-exchange - single-move combination as the improvement heuristic performed better than other combinations. All combinations performed better in general than the standard Branch and Cut technique in terms of solution time.

ACKNOWLEDGMENTS

The author thanks his advisor Dr. Halit Uster, Department of Industrial Engineering, Texas A&M University, for his helpful guidance and educational contributions. The author also thanks Mr. Vishal Karnik M.S., Department of Industrial Engineering, Texas A&M University, for providing valuable input during the implementation of the model using CPLEX.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Telecommunication networks form an integral part of life. The links and nodes which form the backbone of such networks however, are susceptible to failures. These failures create a disruption in the flow of data which could be unacceptable for the user. Considering the wide usage of telecommunication networks, resilience, i.e. the ability of the network to quickly recover from a failure, has become a key element in the design of these networks in order to ensure user satisfaction and quality service.

As a matter of fact, avoiding failures on networks is not always possible. Designing network structures that survive these failures have become the order of the day. If a network structure, with spare or excess capacity, is able to remove interrupted traffic by the failure of some its elements, then the network is said to be survivable (Soriano et al., 1998). Also, if the rerouting of traffic is automated and the network reconfigures itself in case of failure, then the network is said to be self-healing (Soriano et al.,1998). With the introduction of SONET (Synchronous Optical Network) technology, rings have become the preferred survivable network structure. The SONET network planning is usually performed in 3 stages:(1) Logical Ring Design (2) Mapping the logical rings onto the existing fiber optic network (3) Connecting the rings and routing the traffic effectively. The set of nodes in the network are optimally grouped into rings with each node being a part of a ring, and each ring being connected to at least one other ring at two or more nodes. This network structure ensures faster recovery from a single link or single node failure (Laguna, 1994; Luss et al., 1998; Soriano et al., 1998).

This thesis follows the style of European Journal of Operational Research.

A network structure that incorporates rings (cycles connecting a set of terminals or central locations given as nodes) can be designed in two stages. In the first stage, a set of disjoint rings (each node being a part of a single ring) is formed in such a way that the total link length of cycles is minimized, and in the second stage, these disjoint rings are connected via another main ring. In both stages, several different criteria can be considered depending on the design requirements. With this kind of application in mind, we focus on the first stage problem with a node number balance criterion among the rings. Specifically, given a set of nodes and distances between them, the Balanced Disjoint Rings (BDR) Problem is the minimum total link length clustering of nodes into a given number of disjoint rings in such a way that there is almost the same number of nodes in each ring.

The BDR problem is related to the well-known traveling salesman problem (TSP) in which, given a set of nodes and distances, one seeks to find the minimum length tour that visits each node exactly once. In BDR, if the balance requirement among the rings is relaxed and the number of rings required is set to one, then we obtain the TSP. It can easily be argued that the BDR problem is also NP-hard, and hence we do not expect to obtain a polynomial time algorithm for its solution.

Therefore, we are motivated to develop an efficient heuristic solution approach, which finds close to optimal solutions in reasonable amount of time. In general, a neighborhood search based heuristic algorithm starts with an initial solution and utilizes a neighborhood function to generate improving solutions around the current solution. The algorithm accepts the new neighborhood solution if it improves the current best objective value. This process of finding the neighborhood solutions, replacing the current solution with a neighborhood solution that has a better objective value is repeated until a termination criterion is reached.

During the design of a heuristic, the decision about the structure of the neighbor-

hood function is of strategic importance. As the size of the neighborhood increases, the longer it takes to solve the problem resulting in fewer iterations per unit time (Ahuja et al., 2000). Hence, to develop accurate solutions with lesser number of iterations, we need to search the large neighborhood in an efficient manner. In this study, a solution construction heuristic and a solution improvement heuristic with a multi-exchange neighborhood generation and search procedure (based on very large scale neighborhood search) will be considered. The technique produces improved neighborhood solutions without enumerating and evaluating all neighbors in the large neighborhood. The results obtained by the standard Branch and Bound Technique and the multi-exchange heuristic approach for randomly generated problem instances will then be compared.

I.1. Motivation and Objective

An analytical model to help design good survivable networks and effective solution methodologies to solve these problems could be of immense help to the telecommunication industry. We will formulate our problem of interest that has applications in this area. One specific feature of the model is that it incorporates constraints for a balanced distribution of load across the network. This ensures that the rings in the network would have a uniform load pattern. We develop effective solution methodologies and test their efficiency via an experimental set-up that considers parameter values varying both in problem input data and solution procedure parameters. This experimentation helps us to observe the trade offs one would have to make while solving such difficult problems.

The objectives of this research are to (i) formulate a mathematical model for the Balanced Disjoint Ring problem, (ii) provide heuristic solution methodologies

that performs better than the standard Branch-and-Cut technique that can be implemented using powerful readily available software such as CPLEX 9.0[1].

I.2. Organization of the Thesis

This thesis is structured as follows. The next section gives the literature review. Chapter II gives the notation, definition and problem formulation. Following that, in chapter III, we present various heuristic solution procedures to solve our model efficiently. Chapter IV provides computational results for comparison of exact and heuristics methods, and finally, in chapter V, conclusion and recommendations for future research are discussed.

I.3. Related Literature

This chapter provides a summary of the literature in two areas with respect to the problem in hand. The literature related to the problem definitions and solution methodologies similar to the BDR problem are summarized.

The BDR is related to the well-known TSP. TSP has several variations and one of them, multiple TSP (m-TSP), generates multiple cycles similar to BDR. In m-TSP, a given number of salesmen are located at a specified base node and the problem is to find a tour for each salesman to cover a portion of a given graph so that each node of the graph is visited only once.Solution approaches to m-TSP was developed by Bellmore and Hong (1974). They transformed the m-TSP to a standard TSP for which standard algorithms are available. They represented the system as an expanded graph with $(m-1)$ more nodes than the original graph. Hong and Padberg (1977) showed that symmetric m-TSP (symmetric cost or distance structure) is equivalent to

[1]CPLEX is a trademark of ILOG, Inc.

the standard symmetric TSP involving $n+m+4$ cities (n is the number of cities). In addition to this, Jonker and Volgenat (1986) presented an improved transformation for the undirected single depot m-TSP to symmetric TSP by making m-1 copies of the depot. The results were also computationally verified. Gavish and Srikanth (1986) formulated the problem as integer linear program with fixed and variable number of salesmen and provided an efficient branch-and-bound scheme with lagrangian bounds. More information about the TSP variants could be found in Gutin and Punnen (2002).

Another related problem is the multi-depot location-routing problem (MDLRP). MDLRP involves solving a facility location-allocation problem (i.e., identifying open depot locations from the set of candidate locations and assigning customer demands to these depots) and vehicle routing problem (routing of vehicles from the depots to the assigned customers) simultaneously. Perl and Daskin (1983) proposed a heuristic approach to solve the problem by decomposing the original problem to subproblems. Another heuristic approach for the same problem was provided by Hansen et al. (1994). Wu et al. (2002) developed a heuristic method to solve the multi depot location routing problem. They decomposed the problem into Location-Allocation and Vehicle Routing Problems. Some approximate algorithms for solving the MDLRP are described in the literature (Laporte, 1988; Madsen, 1983; Jacobsen and Madsen, 1980; Srivastava, 1993; Tuzun and Burke, 1999; Lin et al., 2002; Salhi and Rand, 1989; Min et al., 1998).

The most related study to the problem under consideration found in the literature is the Hamiltonian p-median problem (HPMP) which is obtained by combining the p-median location and TSP problems (Branco and Coelho, 1990). In a typical p-median problem, given a set of candidate locations and customer nodes, we are interested in locating p new facilities and assigning the customers to these facilities to be served directly. The objective is to minimize the total transportation cost. The

only difference between p-median and HPMP is that, in HPMP, the customers are not served directly. There is a route for each new facility and the customers on a route are served by the corresponding facility. Branco and Coelho (1990) provided two formulations for this problem. The HPMP is also an NP-hard problem. Branco and Coelho (1990) provided four heuristics to solve HPMP. The clustering heuristic is a construction heuristic. It selects the seed nodes based on the farthest distance and groups the remaining nodes under the seed nodes to form the clusters. The TSP is solved for each cluster. The 3-optimal method starts with an initial feasible solution and tries to find an improving solution by changing three links systematically. The shrinking heuristic starts by solving the 2- matching problem and then shrinks the solution to obtain a feasible solution. 3-optimal algorithm is applied to this solution. The authors also provided details about the spanning walk heuristic. Glaab and Pott (2000) considered a similar version of the problem on a directed graph, providing good, but rather limited polyhedral results without any valid inequalities that can be employed.

In terms of the solution methodology, literature related to construction and improvement heuristics are summarized. The clustering heuristic given by Branco and Coelho (1990) relates closely to the construction heuristic requirements for the BDR problem. Multi-Exchange heuristic is a neighborhood search based improvement heuristic. Multi-Exchange falls into the class of the very large scale neighborhood search (VLSN) algorithms and is based on the cyclic transfer of nodes between rings. Cyclic exchange produces more neighbors than the conventional two-exchange neighborhood where assignments of pairs of nodes are exchanged between two rings. The theory of cyclic transfer was first introduced by Thompson and Orlin (1989). The authors discuss assignment of nodes to clusters and configuration of nodes within clusters for various well-known problems such as facility location and vehicle rout-

ing problems. A network optimization based cyclic exchange neighborhood search methodology was described by Ahuja et al. (2000) with specific application to the capacitated minimum spanning tree problem. For this problem, Ahuja et al. (2003) introduced a better performing composite VLSN structure where there is at least one node subject to the cyclic transfers between subtrees. Thompson and Psaraftis (1993) investigated the application of cyclic transfer algorithms for multi-vehicle routing and scheduling problems. They analyzed the worst case performance for various problem instances and proposed a computationally efficient method for finding negative cost (the ones providing improvement over the current solution) cyclic transfers. Ahuja et al. (2004) explored the neighborhood induced by customer multi-exchanges and facility moves for the single source capacitated facility location problem. Finally, Ahuja et al. (2002) provided a detailed survey about the following neighborhoods search techniques used in VLSN search algorithms: (1) Variable Depth Method (2) Large neighborhood searched using network flow techniques or dynamic programming, and (3) large neighborhoods generated by restrictions of the original problem.

CHAPTER II

PROBLEM DEFINITION AND FORMULATION

We first define the notation used in the mathematical formulation of BDR. Let $G = (N, E)$ be a complete graph with a node set of N and edge set of E where the length of edge (i, j) is δ_{ij}. Let n be the number of nodes, and c be the number of disjoint rings to be formed. We introduce a binary variable x_{ijk} that represents the assignment of edges to rings, i.e., $x_{ijk} = 1$ if edge (i, j) is in ring k for $i, j = 1, \ldots, n$ and $k = 1, \ldots, c$, and $x_{ijk} = 0$, otherwise. Additionally, we introduce a set of binary variables y_{ik} for the assignment of nodes to rings, i.e., $y_{ik} = 1$ if node i is assigned to ring k for $i = 1, \ldots, n$ and $k = 1, \ldots, c$, and $y_{ik} = 0$, otherwise. Let N_k be the set of nodes in ring k; S_k be a nonempty subset of N_k; and K be the index set for rings, $K = \{1, \ldots, c\}$. Then,

$$\text{Min} \qquad \sum_{k=1}^{c} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij}\, x_{ijk} \tag{2.1}$$

subject to

$$\sum_{j=1}^{v-1} x_{jvk} + \sum_{j=v+1}^{n} x_{vjk} = 2\, y_{vk} \quad \forall\, v \in N, \forall\, k \in K, \tag{2.2}$$

$$\sum_{i=1}^{n} y_{ik} \geq \lfloor n/c \rfloor - 1 \quad \forall\, k \in K, \tag{2.3}$$

$$\sum_{i=1}^{n} y_{ik} \leq \lfloor n/c \rfloor + 1 \quad \forall\, k \in K, \tag{2.4}$$

$$\sum_{k=1}^{c} y_{ik} = 1 \quad \forall\, i \in N, \tag{2.5}$$

$$\sum_{i \in S_k} \sum_{\substack{j \in S_k \\ j > i}} x_{ijk} \leq\ (|S_k| - 1)\, y_{vk} + |S_k|\,(1 - y_{vk}) \quad \forall S_k,\ \forall\, v \in N \setminus S_k,\ \forall\, k \in K, \tag{2.6}$$

$$x_{ijk}, y_{ik} \in \{0, 1\} \quad \forall\, i, j \in N, \forall\, k \in K. \tag{2.7}$$

The objective function (2.1) minimizes the total edge length over the rings. Constraints (2.2) state that each node has a degree of 2, this is required for the proper formation of rings. We impose the constraints (2.3) and (2.4) in order to obtain a balanced formation of disjoint rings. The left hand side of these constraints represents the number of nodes assigned to a ring k. On the right hand side, $\lfloor n/c \rfloor$ represents the rounded down value of the average number of nodes per ring in a completely balanced design. Therefore, we require the number of nodes in any ring to be within ∓ 1 range of this integral average node count. Constraints (2.5) guarantees that each node is assigned to exactly one ring. Finally, constraints (2.6) are for elimination of sub-rings within rings. Specifically, these constraints eliminate the situations such as the one depicted in Figure 1 which shows a possible solution to a problem required to form two disjoint rings. The nodes 1, 2, 3, 4, 5 and 6 are considered to be a part of the ring $R1$ and they actually form two sub-rings in $R1$. A sub-ring elimination constraint that would avoid this particular situation can be obtained as follows: Given that $N_1 = \{1, 2, 3, 4, 5, 6\}$ assume a subset selection as $S_1 = \{1, 2, 3\}$ and a node in $N \setminus S_1$ as $v = 4$ so that $x_{121}, x_{131}, x_{231}, y_{41} = 1$. Then, the constraint $x_{121} + x_{131} + x_{231} \leq (|S_1| - 1)\, y_{41} + |S_1|\,(1 - y_{41}) \Longrightarrow 3 \leq 2$ is clearly violated.

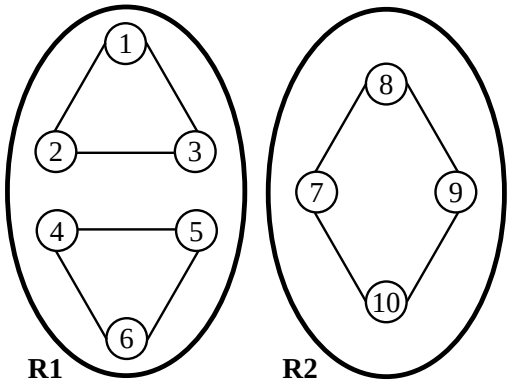

Figure 1 Sub-Rings in Solution

CHAPTER III

SOLUTION APPROACHES WITH LOCAL SEARCH

A neighborhood function is an important ingredient of a local search algorithm. Given a feasible solution s, a neighborhood function defines a mechanism to generate other feasible solutions, $\mathcal{N}(s)$. The solutions contained in $\mathcal{N}(s)$ are called the neighboring solutions of s. Clearly, the set $\mathcal{N}(s)$ is a subset of the complete feasible solution space to the problem at hand. We start each iteration of a local search improvement algorithm with a feasible solution, whose goodness value (this is usually its associated objective value) is calculated, and generate its neighborhood solutions. We pick the neighboring solution with the best goodness value if it is better than the current best solution and start the second iteration with this new improved solution. We continue the iterations until there is no improving solutions in the neighborhood of the current best solution. The initial feasible solution in a local search is obtained via a construction heuristic. Further details and various approaches to local search can be found in (Aarts and Lenstra, 1997; Michalewicz and Fogel, 2000).

If the neighborhood function is able to generate a large set of neighboring solutions we have a better chance of obtaining good solutions, i.e., the solution space is more explored. However, if an efficient neighborhood generation method is not available, the gain in solution quality comes at the expense of increased run time of the algorithm. On the other hand, if we employ a neighborhood function that involves less exploration and thus provide faster run times, this gain comes at the possibility of not obtaining acceptably good solutions. That is, in general there is a trade off between the solution quality and solution time when local search heuristics are used.

In this thesis, we investigate three *alternative* construction heuristics; and two neighborhood functions that are to be used *simultaneously* in a local search frame-

work. The first neighborhood function is multi-exchange neighborhood which generates a relatively large number of neighboring solutions. However, efficient techniques exist for the generation of these solutions. The second one is a simple single-move neighborhood, although generates a relatively small set of neighboring solutions, it provides an opportunity for diversification in the solution space explored by the multi-exchange. We give the details of the construction heuristics and the improvement algorithm employing the neighborhood functions below. Since they will be used in the subsequent discussions, we first review the related standard definitions from graph theory. For a general graph $G = (N, E)$ we define the following:

Path: A path between vertices s and t, (s,t)-path, is a sequence of arcs of the form $(s, i_1), (i_1, i_2), \ldots, (i_{k-1}, (i_k), (i_k, t)$ where $\{s, t, i_1, \ldots, i_k\}$ are non-repeated (distinct) vertices in the graph.

Cycle: A cycle containing a vertex s is the augmentation of an (s,t)-path and the (t,s) arc. Informally, a cycle is a sequence of vertices that forms a path and additionally the end vertex of the path is connected to the start one. In a general graph, a cycle must contain at least three arcs.

Tree: A tree is a graph in which any two vertices are connected by exactly one path.

Spanning Tree: A spanning tree is a subgraph of G which is a tree and connects all the vertices. A minimum spanning tree (MST) or minimum weight (total arc cost) spanning tree of G is then a spanning tree with weight less than or equal to the weight of every other spanning tree of G.

Travelling Salesman Problem (TSP): Finding the minimum weight cycle in G that visits all the vertices. We represent the objective function value of a TSP solution on G as $t(G)$.

III.1. Evaluating a Solution

In general, the quality of a solution is represented by the value of the objective function or the cost it implies. The cost for the BDR problem is given by the sum of the lengths of the links constituting the rings and the overall objective is to minimize this cost. Let the set of nodes N be partitioned into c rings $\mathcal{P} = \{N_1, N_2, \ldots, N_c\}$, where $N_i, i = 1, \ldots, c$, represents the set of nodes in a ring i and $|N_i|$ satisfies the constraints (2.3) and (2.4) on allowable number of nodes in a ring. A given partitioning $\mathcal{P}$, in which $\bigcap_{i=1,\ldots,c} N_i = \emptyset$ and $\bigcup_{i=1,\ldots,c} N_i = N$, identifies a solution to our problem. Let the associated cost of the partitioning $\mathcal{P}$ be $Z(\mathcal{P})$. $Z(\mathcal{P})$ is determined by solving a TSP in each ring and then summing up the link lengths. We solve the TSP in each ring i on the graph induced by its set of nodes N_i using the Christofides' TSP heuristic (Christofides, 1976). The Christofides' heuristic is a 3/2-approximation algorithm, i.e., it finds solutions whose objective values are at most 50% worse than the optimum value. Our reason to employ the Christofides' heuristic is that the cost evaluations are heavily used in the course of the algorithm mostly to compare the solutions in a neighborhood and it provides the means of doing this quickly and effectively. However, we also consider finding optimum TSP tours in rings once a final solution is reached.

III.2. Multi-Exchange Neighborhood

Various very large scale neighborhood functions are given by Ahuja et al. (2002). For our problem, a multi-exchange neighboring solution can be obtained by shifting a subset of nodes on a cycle that contains exactly one node from the rings that it visits. Each ring is visited at most once on such a cycle, and thus, it is called a subset-disjoint cycle. Let the number of visited rings on a subset-disjoint cycle be V, $V \leq c$, and let

n_i be a node in N_i for $i = 1, \ldots, c$. Given $\mathcal{P}$, an example subset-disjoint cycle of size V can be given as $n_1 - n_2 - n_3 - \ldots - n_V - n_1$ and it represents the multi-exchange operation in which the node n_1 moves from ring N_1 to N_2, the node n_2 moves from ring N_2 to N_3 and so on until the node n_V moves from ring N_V to N_1. Although $c-V$ partitions are not affected with this cyclic change, in general, this results in a new partition of the nodes with the following rings $\mathcal{P}' = \{N_1', N_2', \ldots, N_V', \ldots, N_c\}$ and associated cost of $Z(\mathcal{P}')$. The cost of this exchange is given by $Z(\mathcal{P}') - Z(\mathcal{P})$. If this difference is negative then the exchange cycle is called "negative cost subset-disjoint cycle".

For the given partition $\mathcal{P}$, $\mathcal{N}(\mathcal{P})$ is the set of neighboring solutions obtained by the subset-disjoint cyclic exchanges. This can yield a very large neighborhood. Assuming that n/c is an integer and each ring includes the same number of nodes the cyclic exchange neighborhood size is $c!\,(n/c)^c$ which is of order n^c. As c, the number of rings included in the exchange, increases the neighborhood size increases exponentially. Explicitly evaluating all such possible subset-disjoint cycles would be computationally prohibitive. However, the improvement graph described below, could be used to implicitly search the neighborhood.

An improvement graph $I(\mathcal{P})$ is a directed graph that can be generated for a given solution $\mathcal{P}$. The vertices of $I(\mathcal{P})$ are given by N and the directed arcs are constructed between every pair of nodes that are in different rings N_i, $i = 1, \ldots, c$. Each arc (i, j) on $I(\mathcal{P})$ represents a feasible move of node n_i from its current ring(N_i) to the ring containing the node $n_j(N_j)$. The node n_j subsequently leaves the ring N_j. The cost on a such a directed arc (i, j) is given by $t(N_j \setminus \{j\} \cup \{i\}) - t(N_j)$, i.e., the change in the TSP value of ring containing n_j after n_j is excluded and n_i is included in that ring. A subset-disjoint cycle on $I(\mathcal{P})$ with a negative total link cost is called a negative cost subset-disjoint cycle, and henceforth such cycles will also be referred

to as valid cycles since they represent the cyclic exchanges that provide improvement over $Z(\mathcal{P})$. Similarly, the negative cost subset-disjoint paths will be referred to as valid paths.

It is easily observed that directed cycles can be obtained by joining the end points of the directed paths and arcs suitably. For example, the directed cycle $n_1 - n_2 - n_3 - n_4 - n_1$ could be obtained by combining the path $n_1 - n_2 - n_3 - n_4$ and arc $n_4 - n_1$. Hence, all directed cycles could be enumerated by enumerating all directed paths in the network. However, enumeration of all possible paths can be very time-consuming. In order to generate paths efficiently, we utilize a result by Lin and Kernighan (1973) which is also utilized by Ahuja et al. (2003) in solving capacitated minimum spanning tree problem. This result is given in Lemma 1.

Lemma 1 *If* $W = i_1 - i_2 - \ldots - i_c - i_1$ *is a negative cost (directed) cycle, then there exists a node* i_h *in the cycle so that each partial (directed) path* $i_h - i_{h+1}; i_h - i_{h+1} - i_{h+2}; \ldots; i_h - i_{h+1} - i_{h+2} - \ldots - i_{h+k-1}$ *(where indexes are modulo c) is a negative cost (directed) path.*

Lemma 1 implies that for the valid cycle $n_1 - n_2 - n_3 - n_4 - n_1$, there exists a node in the cycle say n_2, such that the paths $n_2 - n_3$, $n_2 - n_3 - n_4$, and $n_2 - n_3 - n_4 - n_1$ are all valid paths. This result gives us a way to enumerate all valid cycles by enumerating the valid paths in the network. While enumerating the valid cycles, the nonnegative cost paths are ignored. This also effectively helps us to avoid the generation of identical cycles in terms of their formation such as the directed cycles $n_1 - n_2 - n_3 - n_4 - n_1$ and $n_3 - n_4 - n_1 - n_2 - n_3$. In addition, the enumeration is further improved by eliminating dominated paths which will be explained in section III.5.1.

The figures provide a better understanding of the problem and solution methodology. Figure 2 provides the geographical distribution of the nodes and an initial

solution for the problem which could be constructed using an initial construction heuristic. Figure 3 shows a valid cycle that could be generated with this initial solution. Figure 4 illustrates the solution after the implementation of the valid cycle.

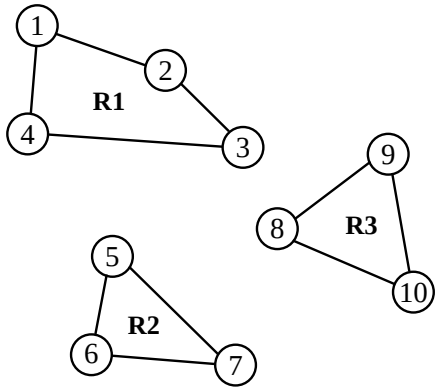

Figure 2 Initial Ring Structure - Geographical Interpretation

III.3. Single Move Neighborhood

In single move neighborhood function, unlike multi-exchange, a node is transferred to another ring without any node leaving the destination ring. Assuming the same problem setting as explained in the previous section, $n_1 - n_2$ represents the transfer of node n_1 from N_1 to the ring n_2 belongs, i.e., N_2. This results in a new partition $\mathcal{P}' = \{N_1', N_2', N_3 \ldots, N_c\}$ with the associated cost of $Z(\mathcal{P}')$. This single move is accepted as a move that improves the current objective value of the solution if the cost change $Z(\mathcal{P}') - Z(\mathcal{P})$ is negative. Figure 5 depicts a possible move that could exist in the solution after the implementation of the valid cycle. Figures 6 represents the solution after the implementation of the move.

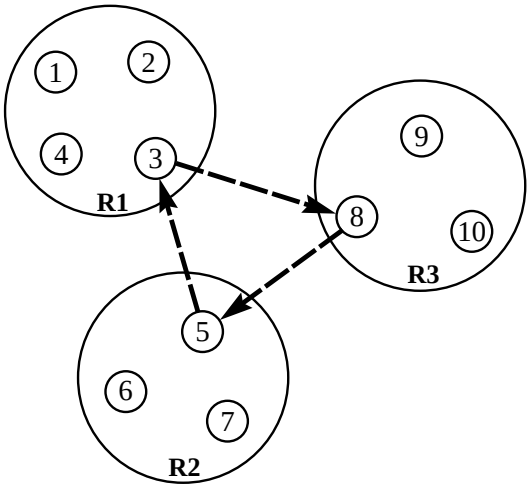

Figure 3 Multi-Exchange - Valid Cycle ($n_3 - n_8 - n_5 - n_3$)

III.4. Construction Heuristics

In a construction heuristic for our problem, the given set of nodes have to be partitioned into sets based on the number of rings that are required. Also, the partitioning should be done in such a way that each partitioned set has almost the same number of nodes. With n being the number of nodes and c the number of rings, n/c represents the average number of nodes per ring. However, this is not always an integer value. The balanced distribution of nodes in each ring is then implemented by taking the floor value of n/c and then introducing the range of $\lfloor n/c \rfloor - 1$ (lower bound) to $\lfloor n/c \rfloor + 1$ (upper bound). This range determines the number of nodes allowed in each ring and as the range between upper and lower bound is two, the rings are forced to have almost the same number of nodes. To start the improvement heuristic, one

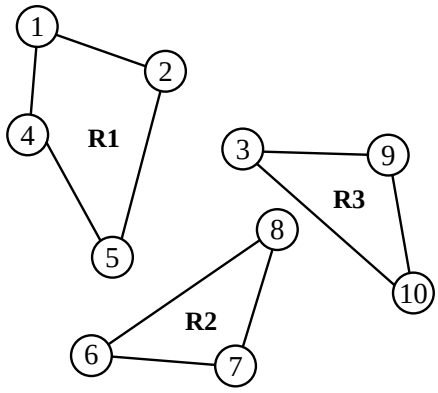

Figure 4 Ring Structure after Exchange - Geographical Interpretation

such feasible solution needs to be generated. We develop the following approaches to generate the initial solution.

III.4.1. Break -MST Method

Break MST method uses the Kruskal's algorithm to form the minimum spanning tree (MST) (Ahuja et al., 1993). With the resulting MST, the required number of partitioned set of nodes are formed by breaking the links on the MST in decreasing order of their link lengths. The number of links that are broken is one less than the number of required rings. In figure 7, the Minimum spanning tree is formed for the whole problem using the Kruskal's algorithm in step 0. In step 1, the end nodes(LinkBegin, LinkEnd) of the longest link is found. This link is excluded from the MST solution and the corresponding distance entry is set to zero in step 3. In step 4, with the broken MST solution, partitions are formed. If the number of partitions is

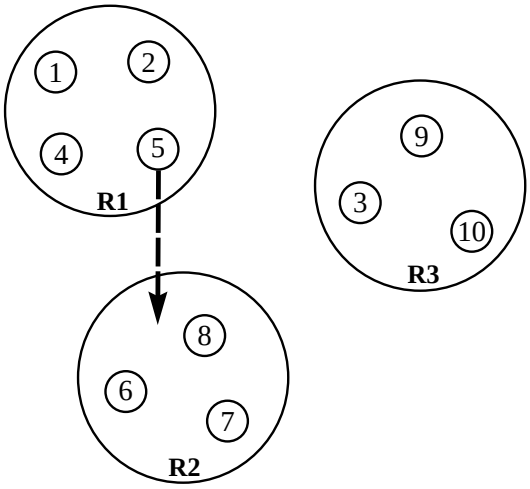

Figure 5 A Single Move

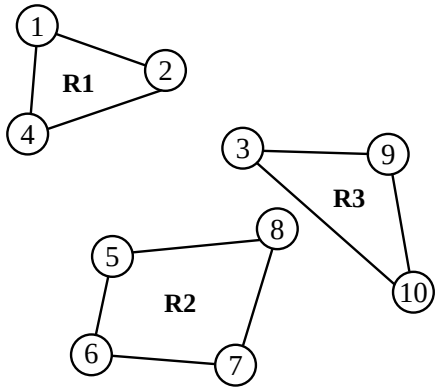

Figure 6 Solution after Move - Geographical Interpretation

less that the number of disjoint rings that are required, then the algorithm goes back to step 2 to break more links in the decreasing order of their lengths. This algorithm terminates when the number of partitions is equal to the number of rings. However, this method does not always yield a balanced partition of nodes. This drawback motivated the following two construction heuristics that would yield solutions with a balanced partition of nodes. The second one of these next two heuristics utilizes the Break-MST method. The overall algorithm for the Break-MST method is given in figure 7. The parameters provided within the parenthesis following the function names in pseudo-codes represent the important inputs for the function.

Step 0: mst_sol = $MST\_kruskal$(Problem_Data)
Step 1: [LinkBegin, LinkEnd] = $Find\_Maximum\_Link$(mst_sol)
Step 2: $Break\_Link$(mst_sol, LinkBegin, LinkEnd)
Step 3: new_mst_sol = Update mst_sol
Step 4: Partition_Sets = $Make\_Partition$(new_mst_sol)
if ($Partition\_Sets.Size \neq Numberof Rings$) **then**
  Step 5: mst_sol = new_mst_sol
  Step 6: GOTO Step 1
end if
Step 7: Return Partition_Sets

Figure 7 Pseudo-code of the Break MST Construction Heuristic

III.4.2. Distance Method (DM)

In the first phase of the Distance Method we determine a set of "seed nodes", each representing a required ring, i.e., the number of seed nodes is equal to c. We choose the seed nodes in an incremental way. We first find the longest link in the distance matrix and assign its end nodes as the initial two seed nodes. The next seed node is determined such that the sum of distance between this node and the seed nodes

is maximum. This seed node is added to the set of seed nodes and this process is repeated until the number of seed nodes determined is equal to the number of required rings. Each seed node is assigned to a partition set N_i, $i = 1, \ldots, c$, representing the nodes in each ring. The nodes that are not yet included to a partition are called "free nodes", and in the second phase of the heuristic, we assign the free nodes to created partitions based on the distance between the free nodes and the seed nodes. Since the partitions are required to be balanced we assign free nodes to partitions in a cyclic manner, selecting the seed nodes in sequence, one by one, and finding the closest free node to the seed node. The free nodes are assigned to the seed nodes in this fashion. Distance method is similar to the method discussed by Branco and Coelho (1990), however since there is no balancing constraints in (Branco and Coelho, 1990), the assignment are made on a free node basis where a free node is assigned to its closest seed nodes. The overall algorithm for the Distance method is given in figure 8.

III.4.3. Hybrid Method (HM)

Hybrid method combines the above two methods. As the first step in this method, the MST solution is formed by Kruskal's algorithm. We pay special attention while breaking the links to obtain partitions. In particular, the links are broken in the decreasing order of their link lengths. Breaking a link yields two partitions. The first partition is checked to validate if the number of nodes in the partition is less than or equal to the allowed upper bound ($\lfloor n/c \rfloor + 1$). If this condition is satisfied, it is made as one of the required partition set for the formation of rings and a new sorted list of link lengths in the remaining MST is created. If the first partition does not satisfy the condition, then the second partition is checked for the same condition. If breaking the longest link does not yield any partition that satisfies the upper bound condition, the broken link is rejoined and the next link in the sorted list is chosen.

Step 0: Distance_Matrix = $Create\_Distance\_Matrix$(Problem_Data)
Step 1: [LinkBegin, LinkEnd] = Max_Link_Length(Dist_Matrix)
Step 2: Update Distance_Matrix
(i.e Set the Distance_Matrix[LinkBegin][LinkEnd] = 0)
Step 3: Partition_Set[1] = {LinkBegin}
Step 4: Partition_Set[2] = {LinkEnd}
Step 5: Seed_Set = {Partition_Set[1],Partition_Set[2]}
Step 6: Temp_set = {LinkBegin, LinkEnd}
for i=3 : Number_of_Rings **do**
Step 7: Seed_Node = Find the node that is farthest apart from Temp_set
Step 8: Partition_Set[i] = {Seed_Node}
Step 9: Add Seed_Node to Temp_Set
Step 10: Add Partition_Set[i] to Seed_set
Step 11: Update Distance_Matrix
end for
Step 12: i=1
Step 13: $Number\_of\_iterations = Number\_of\_Nodes - Number\_of\_Rings$
while Number_of_iterations > 0 **do**
if ($i = Temp\_set.size$) **then**
Step 14: i=1;
end if
for i=1 : Temp_set.size **do**
Step 15: Free_node = Find the node at minimum distance from Temp_Set[i]
Step 16: Add Free_node to Partition_Set[i]
Step 17: $Number\_of\_iterations = Number\_of\_iterations - 1$
Step 18 :Update Distance_Matrix
Step 19: $i = i + 1$
end for
end while
Step 20: Return Partition_Sets

Figure 8 Pseudo-code of the Distance Method Based Construction Heuristic

This is repeated until the number of links broken is equal to the number of rings that are required for the problem.

Once the initial partition is obtained, the distance method is used to assign the remaining nodes (free nodes) to the initial partition sets created in the previous step. The assignment of a free node to the partition is decided based on the minimum distance between this free node and the nodes already assigned to the partition. Similar to the Distance Method, for each partition we find the free node that is closest to its node set in a cyclic manner until no free nodes are left. While assigning the free nodes, care is taken so that the number of nodes in each partition set does not exceed the upper bound.

After all the nodes have been assigned, the lower bound ($\lfloor n/c \rfloor$ - 1) condition is validated for all partitions. If the number of nodes in any partition is less than the lower bound, the nodes are arranged suitably among the rings (partitions) so that the lower bound and upper bound conditions are simultaneously satisfied. This is done by transferring the nodes from rings with number of nodes more than the lower bound to the rings that need nodes to satisfy the lower bound condition. The overall algorithm for the Hybrid method is given in figure 9.

```
Step 0: Distance_Matrix = Create_Distance_Matrix(Problem_Data)
Step 1: MST_Solution = Find_MST(Distance_Matrix)
Step 2: Sorted_List = Sort_Descending_Link_Lengths(MST_Solution)
Step 3: Upper_Bound = ⌊Number_of_nodes/number_of_rings⌋ + 1
Step 4: i=1
START:
if (i <= Number_of_Rings) then
  for j=1 : Sorted_List.size do
    Step 5: [LinkBegin, LinkEnd] = Sorted_List[j]
    Step 6: [Set1, Set2] = Break the link
    Step 7: Update MST_Solution ... (Set the link to zero)
    Step 8: Update Dist_Matrix ... (set Dist_Matrix[LinkBegin][LinkEnd] = 0)
    if (Set1.size < Upper_Bound) then
      Step 9: Partition_Set[i] = Set1
      Step 10: i = i + 1
      Step 11: Update Distance_Matrix, MST_Solution
      Step 12: Sorted_List = Sort_Descending_Link_Lengths(MST_Solution)
      Step 13: GOTO START
    else
      if (Set2.size < Upper_Bound) then
        Step 14: Partition_Set[i] = Set2
        Step 15: i = i + 1
        Step 16: Update Distance_Matrix, MST_Solution
        Step 17: Sorted_List = Sort_Descending_Link_Lengths(MST_Solution)
        Step 18: GOTO START
      else
        Step 19: j = j + 1
        Step 20: Rejoin the broken link
      end if
    end if
  end for
end if
```

Figure 9 Pseudo-code of the Hybrid Method Based Construction Heuristic

Step 21: Temp_set =
{Partition_Set[1],Partition_Set[2],. . . ,Partition_Set[Number_of_Rings]}
Step 22: i=1
Step 23: Number_of_iterations = Number_of_Nodes - Total number of nodes belonging to each Partition Set put together
while Number_of_iterations > 0 **do**
  if $(i = Temp\_set.size)$ **then**
    Step 24: i=1;
  end if
  for i=1 : Temp_set.size **do**
    Step 25: Free_node = Find the node at minimum distance from Temp_Set[i]
    Step 26: Add Free_node to Partition_Set[i]
    Step 27: $Number\_of\_iterations = Number\_of\_iterations - 1$
    Step 28 :Update Distance_Matrix
    Step 29: $i = i + 1$
  end for
end while
Step 30: Return Partition_Set

Figure 9 Continued

III.4.4. Greedy Randomized Adaptive Search Procedure (GRASP) with Distance Method

GRASP framework can be utilized to generalize almost any construction heuristic to a multi-start approach supported by randomization. The details of the overall framework can be found in (Feo and Resende, 1995; Resende and Riberio, 2003). In a general greedy construction heuristic, a feasible solution is generated in such a manner that at each step of the procedure the most benefiting action taken. In that sense, our distance method (DM) is essentially a greedy method where the seed nodes pick free nodes to include in their partition in a greedy fashion. The first generalization that GRASP brings is the randomization. For this purpose, instead of picking the best free node for a partition, we generate *cd* number of best (again based on proximity) free nodes and pick one of them randomly, where each has a probability of $1/cd$ to

join the partition under consideration. The second generalization is the adaptiveness which refers to modifying the data used to take actions at iterations so that the partial solution obtained in the process is accounted for. For this purpose, we modify the distance method in such a way that at each iteration while picking a free node for a partition, we do not only consider the proximity of the free node to the seed node, but its proximity to the existing set of nodes in that partition. The pseudo-code for the distance method with GRASP is given in figure 10. In summary, at each iteration (that is, a given partition under consideration), for each free node, we first find a representative distance measure d_f where f is a free node and d_f is the distance between f and the node closest to it in the partition. We later pick cd distinct free nodes with the smallest d_f values and randomly choose one of them to be included in that partition. We continue in this fashion, one partition at a time, until all the free nodes are assigned. Towards the end of iterations, if the number of free nodes left is less than the parameter cd we choose a free node in nonrandomized greedy way, but still employing adaptiveness. A usually employed feature of GRASP is the opportunity it provides for efficient multi-start. Since an initial feasible solution is constructed via randomization each time the GRASP is applied it is likely to generate a different initial solution. Thus, we start an improvement algorithm that employs GRASP in the construction phase several times, each improvement routine is started with a different initial solution.

Step 0: Distance_Matrix = $Create\_Distance\_Matrix$(Problem_Data)
Step 1: [LinkBegin, LinkEnd] = Max_Link_Length(Distance_Matrix)
Step 2: Update Distance_Matrix
(i.e Set the Distance_Matrix[LinkBegin][LinkEnd] = 0)
Step 3: Partition_Set[1] = {LinkBegin}
Step 4: Partition_Set[2] = {LinkEnd}
Step 5: Seed_Set = {Partition_Set[1],Partition_Set[2]}
Step 6: Temp_set = {LinkBegin, LinkEnd}
for i=3 : Number_of_Rings **do**
Step 7: Seed_Node = Find the node that is farthest apart from Temp_set
Step 8: Partition_Set[i] = {Seed_Node}
Step 9: Add Seed_Node to Temp_Set
Step 10: Add Partition_Set[i] to Seed_set
Step 11: Update Dist_Matrix
end for
Step 12: i=1
Step 13: $Number\_of\_iterations = Number\_of\_Nodes - Number\_of\_Rings$

Figure 10 Pseudo-code of the GRASP Based Distance Method

while Number_of_iterations > 0 **do**
  for j=1 : Temp_set.size **do**
    if ($i == Number\_of\_Rings$) **then**
      Step 14:i=0;
    end if
    if ($Number\_of\_iterations > Buffer.size$) **then**
      for k=1 : Buffer.size **do**
        Step 15: Free_node = find the node that is minimum to all the nodes in Partition_Set[i]
        Step 16: Add Free_node to Buffer;
        Step 17: Update Distance Matrix(avoids similar nodes in the Buffer)
      end for
      Step 18: Selected_Node = Randomly select from a node from Buffer
      Step 19: Add Selected_Node to Partition_Set[i]
      Step 20: Update Dist_Matrix (the selected node is not picked again)
      Step 21: $Number\_of\_iterations - -$
      Step 22: $i = i + 1$;
      if ($Number\_of\_iterations = 0$) **then**
        Step 23: Return Partition_Set;
      end if
    else
      Step 24: Free_node = find the node that is minimum to all the nodes in Partition_Set[i]
      Step 25: Add Free_node to Buffer;
      Step 26: Update Dist_Matrix (the selected node is not picked again)
      Step 27: $Number\_of\_iterations - -$
      Step 28: $i = i + 1$;
      if ($Number\_of\_iterations == 0$) **then**
        Step 29: Return Partition_Set
      end if
    end if
  end for
end while
Step 30: Return Partition_Set.

Figure 10 Continued

III.5. Complete Heuristic Algorithm

As mentioned before, the complete algorithm has two main components: an initial solution construction heuristic which can be any one of the methods described in the

previous section; and an improvement heuristic which is based on a combined use of multi-exchange and single-move neighborhoods. In this section, we describe the overall solution heuristic algorithm with an emphasis on the improvement components.

The algorithm starts with a random generation of the problem which is used to create the starting initial solution for the Multi-Exchange Heuristic. Improving cycles (valid cycles), as explained above, are detected in the initial solution. The new solution obtained by implementing the improving cyclic exchange is used as input for finding more improving cycles in a following iteration, and this is repeated until there are no improving cycles. This constitutes a multi-exchange improvement component of the algorithm. The obtained solution is then used to find an improving single-move. The best improving move is implemented on the solution and the solution is again used as the starting point for a new multi-exchange improvement routine. The overall algorithm terminates when there are no improving cycles and single-moves detected in succession.

The algorithm is given in figure 11. Step 0 of the algorithm generates a random problem based on the random seed and the number of nodes in the network. In step 1, based on the number of rings required and the problem data, an initial solution, which is a feasible solution to the problem is generated. This initial solution is used as input for the improvement heuristic. In step 3 we create a set of improving cycles (cycle set) that was detected for the current solution and in step 4 and step 5 we make the changes in the solution by implementing the cyclic exchange implied by the best multi-exchange cycle from this cycle set and generate the new best solution and its objective value. This new solution becomes the current solution and used as input for next iteration of the improvement heuristic. Once there are no improving cycles detected on the current solution, the inner while loop is exited. For the current best solution, in step 6, the set of all improving single-moves are detected to form a move

Step 0: Problem_Data =
$Generate\_Problem\_Data$(Random_Seed, Number_of_Nodes)
Step 1: Initial_Solution =
$Create\_Initial\_Solution$(Number_of_Rings, Problem_Data)
Step 2: Solution ← Initial_Solution
Set $Move\_Set.Size = 1$ and $Cycle\_Set.Size = 1$
while Move_Set.Size $\neq 0$ **do**
while Cycle_Set.Size $\neq 0$ **do**
Step 3: Cycle_Set = $Generate\_Improving\_Cycles$(Solution)
Step 4: $Implement\_Multi\_Exchange$(Cycle_Set, Solution)
Step 5: Update_Solution
end while
Step 6: Move_Set = $Generate\_Improving\_Moves$(Solution)
Step 7: $Implement\_Improving\_Moves$(Move_Set, Solution)
Step 8: Update_Solution
end while
Step 8: Return Solution

Figure 11 Pseudo-code of the Complete Heuristic Algorithm

Step 0: arc_set=$Create\_an\_Improvement\_Graph$(Solution)
Step 1: cycle_set=$Create\_Initial\_Cycle\_Set$(arc_set)
for $i = 1 : Number\_of\_Rings$ **do**
  Step 2: cycle_set = $Merge\_Paths$(cycle_set,Number_of_Rings);
end for
if ($cycle\_set.size! = 0$) **then**
  Step 3: Best_Cycle = $Sort\_Cost\_Cycle$(cycle_set);
  Step 4: New_Solution = $Perform\_Exchg$(Best_Cycle,Solution);
end if
Step 5: tsp_tours = $Create\_TSP\_Tours$(New_Solution)
Step 6: Return New_Solution

Figure 12 Pseudo-code of the Multi-Exchange Heuristic

set. In steps 7 and 8, the best improving move from the move set is implemented. The new solution thus obtained is fed as input into the multi exchange heuristic again (Step 3). The whole process terminates when the size of the move set and cycle set are zero i.e., there are no improving moves and cycles detected for the current solution.

III.5.1. The Multi-Exchange Component

In this subsection, we give the details involved in steps 3, 4 and 5 of the complete heuristic algorithm given in figure 11. The idea is to search the neighborhood of the current solution for improving solutions by utilizing valid cycles. The various functions in the enumeration of valid cycles and the algorithm are given in figure 12 and explained subsequently.

III.5.1.1. Creation of Arc Set - Improvement Graph

The set of all arcs that represent the transfer of nodes from one ring to another constitutes the arc set. These arcs and the corresponding nodes, which is the complete set of nodes N, define the improvement graph associated with the current solution. On this graph, arc $(n_i - n_j)$ represents the transfer of node n_i from the ring N_i to N_j and the subsequent removal of n_j from the ring N_j. The associated data structure to represent this data was designed to have the following elements: (*Node1, Node2,* $Arccost_{Node1,Node2}$*, RingFrom, RingTo*). For a pair of nodes (n_i, n_j) the $Arccost_{ij}$ is calculated as $t(N_j \setminus \{n_j\} \cup \{n_i\}) - t(N_j)$ and the elements of the data-structure are (n_i, n_j, $Arccost_{ij}$, N_i, N_j).

III.5.1.2. Creation and Update of a Cycle Set

Initially, the cycle set is equivalent to the arc set, however, in contrast to the arc set, the cycle set changes in the course of the algorithm. For a fixed initial solution and the corresponding improvement graph (arc set), cycle set evolves from containing only the arc set initially to subset-disjoint negative cycles via generation of subset-disjoint negative paths on the improvement graph. Accordingly, we define a new data structure for an entry of cycle set. In general terms, each entry of the cycle set is essentially a path from n_i to n_j. Once the path is closed to form a cycle, this entry is not processed any further.

The data-structure has the following elements: (*path, cost, tail, head, label_set*). The *path* holds the nodes that constitute a valid path. *Cost* gives the savings in the objective value once the valid path is implemented. The *head* and *tail* represents the ending and the starting nodes of the valid path, respectively. The *label_set* keeps track of the rings that are involved in the construction of a valid path. For a directed

path $n_i - n_j - n_k$ on the improvement graph, the elements of the data structure are $([i,j,k],(Arccost_{ij} + Arccost_{jk}), n_i, n_k, \{N_i, N_j, N_k\})$.

In order to update the cycle set in the course of obtaining valid cycles via generation of valid paths we proceed as follows. For each path type entry in the current cycle set (cycles that are already formed are not processed) the set of all paths ($A^{'}$) emanating from its head improvement graph is created. Each of these paths in $A^{'}$ are evaluated for addition to the existing path, a path is only considered for addition if the resulting path still has a negative cost. Furthermore, in the process, we make sure that the *label_set* of the processed path after the addition of the new path does not contain any duplicate elements except the ones corresponding to the *tail* and the *head* once the valid cycle has been formed. If this is the case, this entry becomes a cycle type entry and is not further processed. Otherwise, this check makes sure that the processed path entry stays subset-disjoint. Once a new path entry is obtained, we check if it has any domination relationship with existing paths. If the cost of one path, say (p_1) is less than another path, say (p_2), and the two paths have the same *tail*,*head* and the *label_set*, then P_1 dominates P_2, and hence only p_1 is kept in the cycle set. Finally, the total cost after combining the existing path with a new path should continue to be negative.

If these conditions are satisfied simultaneously, then the new path is combined with the existing path. For example, path $(k - m)$ could be combined with path $i - j - k$ as the head of path $i - j - k$ is same as the tail of $(k - m)$. Assuming that the cost remains negative and the new path is subset-disjoint with existing path in the cycle set, the resulting path becomes $([i,j,k,m],(Arccost_{ij} + Arccost_{jk} + Arccost_{km}), n_i, n_m, \{N_i, N_j, N_k, N_m\})$.

The method is repeated based on the number of rings that are required to be formed to ensure the generation of all the valid cycles. The valid path that is formed

at every iteration of this step finally evolves into valid cycle where the starting and the ending node of the path are the same.

III.5.1.3. Implementation of the Cyclic Exchange

The best valid cycle, i.e., the valid cycle with the most negative cost is selected from the cycle set and implemented. The exchange is done based on the value of *path* in the data structure of the selected valid cycle. After the exchange is implemented, the nodes for each ring are again configured by solving the TSP using the Christofides' heuristic within each ring. As mentioned in the overall heuristic algorithm, the whole process of finding the arc_set, initial improvement graph and cycle set is repeated until no improving cycles (valid cycles) are detected for a given solution. However, we further the search by creating a new configuration of the solution determined till now by detecting improving moves.

III.5.2. The Single-Move Component

The main motivation behind the single-move component is to consider different node number configurations in the rings. Consider a BDR problem on a network with 21 nodes and 3 rings are to be formed. Then, the possible number of nodes in a cycle is (6, 7, 8). Then the possible configurations for 3 rings are (7,7,7), (6,7,8), (6,8,7), (7,6,8), (7,8,6), (8,6,7) and (8,7,6). However, for example, if the initial solution is constructed using the distance method, the solution will surely have a (7,7,7) configuration, and this is very limiting in the search procedure. Since exchanges are performed in a cyclic manner, the initial configuration will never change. To overcome this disadvantage, we devise the single-move component. The formation of move set, the set of all improving moves (valid paths of length one), is very similar to creating the cycle set explained above. The improving move is represented by a node transfer from one ring

to another without any node being removed from the destination ring. This is done by creating the arc set as explained above. With this arc set, the move set is created after checking the following conditions.

- After the move, the origin ring and destination ring should continue to have the number of nodes within the range of $\lfloor n/c \rfloor - 1$ to $\lfloor n/c \rfloor + 1$.
- The number of nodes in each ring should at least be 3 to be able to form rings.
- The move should improve the objective value.

The move set is represented by a data-structure which is the same as the cycle set with the elements (*path, cost, tail, head, label_set*) where the $cost_{ij}$ of moving n_i into the ring n_j belongs is given by $(t(N_j \cup \{i\}) + t(N_i \setminus \{i\})) - (t(N_j) + t(N_i))$ and the elements of the data structure $([i, j], cost_{ij}, n_i, n_j, \{N_i, N_j\})$

The difference between the move set and cycle set is that, unlike the cycle set, the elements of the move set are not combined any further. Once the move set is formed, the best improving move is selected and implemented. The nodes of the rings are again optimally configured by solving a TSP for each ring. With this new solution, we proceed to find improving cycles and the whole procedure is thus repeated again. The algorithm stops when for any given solution there are no improving moves and cycles detected.

CHAPTER IV

COMPUTATIONAL RESULTS

This section details the experimental set up and the results obtained for the BDR problem using CPLEX and the Heuristic for the different test cases. The results are consolidated and average gap and time analysis are done to compare the results obtained by CPLEX and the various combinations of the Construction (Distance, Hybrid and GRASP based Distance method) and Improvement Heuristics (Multi-Exchange and Single-Moves).

IV.1. Experimental Set Up

The node locations are generated in such a way that their coordinates are uniformly distributed between (0,1000). The variations to the problem set is induced by changing the number of nodes in the system. Also, for a specific number of nodes in the system, the number of rings are also varied. Finally, for a specific number of nodes and rings in the system, different problems are generated by varying the random seed value for the random coordinate generator. To be more specific with the parameters, the number of nodes in the system is varied from 15 through 24 in steps of 3. The number of rings is varied from 2 through 5 in steps of 1 and the random seed value for the random coordinate generator is varied between 10 and 100.

For GRASP, *cd* number of best free nodes selected during initial solution construction could be varied. For providing multi-start approach, we generate different initial solutions using GRASP by changing the random seed value used while picking one of the best free nodes selected. Typically, for testing, *cd* was set to 3 and number of multi-starts to the overall heuristic to 3.

IV.2. Exact Solution Methodology Using CPLEX

CPLEX provides optimal solution to the problem using the standard Branch and Cut technique. The formulation for the BDR problem as explained in chapter II, is modeled using CPLEX. While modeling the constraints, constraint (3.3) is excluded initially. The BDR problem is solved without constraint (3.3). The solution provides information about the nodes in each ring and their configuration. The solution is then examined for existence of sub-tours within each ring. The sub-tour detection algorithm, using the configuration and the nodes in each ring as given by the solution, constructs a tour with the nodes in a particular ring. If the number of nodes in the tour is less than the number of nodes claimed to be in that ring as per the solution then the solution has sub-tours. On detection of such sub-tours, constraint 3.3 is added and the problem is solved again. This eliminates the sub-tours in the solution. The parameters are varied as explained in section IV.1 and the final objective value and the solution times are noted.

The CPLEX results, which are optimal, are used as bench mark solutions for the BDR problem. Results obtained from the different heuristic approaches are compared with CPLEX results to determine the quality of solution in terms of the deviation from the optimal and the time taken arrive at the solution.

IV.3. Multi-Exchange Heuristic Results

The heuristic method which was developed with a view to produce good quality solutions with a good trade off in solution time is applied to the same problems that CPLEX solved. The Heuristic method is categorized into

- Distance Method (DM)

- Hybrid Method (HM)

based on the type of construction heuristic used to create the initial solution. For each of these methods, the number of nodes is again varied from 15 through 24 in steps of 3 and the number of rings from 2 to 5 in steps of 1. The random seed value for the random coordinate generator is varied between 10 and 100.

IV.4. DMopt Results

As a small modification to the DM method, the optimal arrangement of the nodes in the final partition sets obtained at the end of the multi-exchange heuristic is done using CPLEX instead of the Christofides' heuristic approach. This is done to test the quality of the partitions generated by the multi-exchange heuristic. From the tables, it could be observed that the deviation of the solutions from the optimal is reduced compared to the DM approach, without much increase in the solution time.

IV.5. GDM Results

The same set of problems solved by DM, are also solved using the GRASP based DM approach (GDM). Solutions thus obtained are further improved by optimally arranging the nodes in the final partition obtained from GDM using CPLEX resulting in the GDMOpt class of solutions. The tables provides detailed information on these results. It could be observed that the deviation of the solutions of GDMOpt from the optimal is reduced compared to the GDM approach, without much increase in the solution time. In GRASP, we also tested shorter valid cycles (less than the number of rings) for multi-exchange neighborhood. For problems with c greater than 3, we terminated the generation of the cycle set when the shortest valid path in the cycle set is 3, instead of c. This modification helps to improve the solution time without

degrading the solution quality.

IV.6. Analyses of Results

The results of CPLEX and the two heuristic methods are compared based on the objective values and solving times. The deviation of the objective value obtained by CPLEX from the heuristic is calculated by the following formula

$$\text{Gap} = 100 * (Heuristic\_Obj\_Value - CPLEX\_Obj\_Value)/CPLEX\_Obj\_Value$$

This is done for all test cases and the average of the deviation for each Node number - Number of rings combination is found.

From Table I, it is observed that the average gap given by DM is better than that given by HM. Also from Table II, it could be noted that the average solving time increases as the number of rings, for specific number of node, increase. From Table III it could be observed that the average gap for DMOpt approach is lower than that obtained by the DM approach. However, the GDM performs the best compared to DM and HM. Distance method solves the problem faster than the Hybrid method. This is contributed by the fact that the Distance method produces better initial solutions (Table IV) than the Hybrid method in most cases. The results of the initial construction heuristic for all test cases are provided in tables V,VI. The GDM approach takes longer to solve the problem because of the multi-start approach that was adopted. Similar result is also observed for GDMOpt and GDM. The heuristic approach, on the whole, performs much better than CPLEX in terms of the solution time. The deviation of the objective value, provided by the heuristic approach, from the optimal is within the acceptable range. The percentage gaps and solution times for the test cases explained in the above sections are summarized in Tables VII,VIII,IX,X,XI,XII,XIII,XIV listed in the Appendix A.

Table I Percentage Gaps between the Heuristics and Optimal Solutions

		DM-Imp		HM-Imp		GDM-Imp	
n	c	Ave.	Max.	Ave.	Max.	Ave.	Max.
15	2	5.18	12.70	14.02	43.18	4.01	9.13
15	3	3.63	9.36	5.56	19.49	2.65	4.50
15	4	2.75	12.24	5.93	21.01	0.98	4.06
15	5	0.00	0.00	0.00	0.00	0.00	0.00
18	2	4.72	11.06	10.30	41.59	4.18	11.06
18	3	6.50	16.14	9.79	26.56	5.10	13.49
18	4	5.97	19.05	5.72	15.12	2.62	9.29
18	5	2.71	12.96	3.92	14.31	1.28	4.87
21	2	6.25	18.20	8.05	24.68	6.07	13.25
21	3	9.62	20.66	11.05	29.11	6.31	20.40
21	4	4.85	9.46	12.42	42.57	3.21	6.40
21	5	8.46	28.76	9.44	28.57	3.61	15.44
24	2	9.50	17.75	8.20	19.92	4.55	10.54
24	3	7.84	25.32	14.76	26.29	7.30	25.32
24	4	6.62	21.25	10.85	30.00	4.47	8.89
24	5	*3.57*	*9.92*	*9.88*	*22.73*	*2.64*	*5.47*

Table II Average Runtimes (Secs) for the Heuristics and Optimal Solutions

n	c	DM-Imp	HM-Imp	GDM-Imp	CPLEX
15	2	2.47	2.90	4.87	21.79
15	3	5.67	5.25	9.63	66.76
15	4	4.58	11.91	15.95	6.41
15	5	15.33	90.97	34.31	21.12
18	2	5.01	7.20	14.76	112.60
18	3	8.35	11.08	34.65	451.48
18	4	11.04	31.93	70.16	84.46
18	5	68.21	441.82	136.97	52.90
21	2	8.42	8.30	38.61	192.86
21	3	19.60	22.67	72.96	1,307.93
21	4	23.00	112.38	225.63	3,903.00
21	5	58.81	1,207.46	588.05	696.06
24	2	15.18	13.83	38.87	1,634.94
24	3	36.07	32.58	170.62	16,147.61
24	4	76.57	246.78	418.97	29,586.14
24	5	*490.71*	*2967.98*	*992.85*	*8,544.10*

Table III Percentage Gaps: DM-Imp, DMOpt-Imp, GDM-Imp, GDMOpt-Imp

		DM-Imp		DMOpt-Imp		GDM-Imp		GDMOpt-Imp	
n	c	Ave.	Max.	Ave.	Max.	Ave.	Max.	Ave.	Max.
15	2	5.18	12.70	2.45	11.30	4.01	9.13	1.78	6.18
15	3	3.63	9.36	1.67	8.05	2.65	4.50	0.69	4.28
15	4	2.75	12.24	1.76	12.21	0.98	4.06	0.04	0.40
15	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
18	2	4.72	11.06	2.05	8.48	4.18	11.06	1.64	8.48
18	3	6.50	16.14	4.67	15.33	5.10	13.49	3.08	10.58
18	4	5.97	19.05	4.71	18.48	2.62	9.29	1.02	5.85
18	5	2.71	12.96	2.01	12.90	1.28	4.87	0.59	3.18
21	2	6.25	18.20	2.78	14.32	6.07	13.25	2.38	12.72
21	3	9.62	20.66	6.84	20.66	6.31	20.40	3.99	20.30
21	4	4.85	9.46	3.39	9.46	3.21	6.40	1.40	4.95
21	5	8.69	28.76	7.82	28.14	3.61	15.44	2.50	14.80
24	2	9.50	17.75	4.62	12.41	4.55	10.54	1.47	4.57
24	3	7.84	25.32	4.70	21.83	7.30	25.32	4.82	21.83
24	4	6.62	21.25	4.57	19.04	4.47	8.89	2.32	7.14
24	5	*3.57*	*9.92*	*1.88*	*9.41*	*2.64*	*5.47*	*0.70*	*3.17*

Table IV Average Percentage Gaps of Construction and Improvement Heuristics

n	c	DM	DM-Imp	HM	HM-Imp
15	2	11.13	5.18	31.06	14.02
15	3	36.68	3.63	48.78	5.56
15	4	29.15	2.75	52.57	5.93
15	5	30.83	0.00	51.14	0.00
18	2	12.43	4.72	32.28	10.30
18	3	32.33	6.50	48.60	9.79
18	4	39.66	5.97	63.42	5.72
18	5	57.24	2.71	72.58	3.92
21	2	15.89	6.25	24.50	8.05
21	3	38.68	9.62	48.74	11.05
21	4	33.51	4.85	64.38	12.42
21	5	51.42	8.46	88.79	9.44
24	2	14.65	9.50	20.63	8.20
24	3	44.53	7.84	49.05	14.76
24	4	47.96	6.62	71.14	10.85
24	5	39.08	3.57	81.95	9.88

CHAPTER V

CONCLUSIONS AND FUTURE WORK

This research focused on development of Balanced Disjoint Rings, which forms the basis for the construction of survivable network structures in the telecommunication industry. The major emphasis was laid upon the development of the mathematical model and solution methodology. The problem, even though is NP hard, was solved using CPLEX solver with concert technology for smaller size problems. From the various test cases it was noted that the solution time and memory usage increased rapidly as the problem size increased. To overcome this problem, heuristic solution methodology, utilizing the "Very Large Scale Neighborhood" concept, was proposed for this problem. This heuristic methodology in addition to using the multi-exchange and single-move neighborhood search methodology, also uses the two construction heuristics, distance method (DM) and hybrid method (HM) to create starting solutions for the heuristic. The results obtained by the heuristics are compared with that of CPLEX. The results clearly indicated that the heuristic solutions are better than that of CPLEX in terms of the average solution time. The average deviation of the heuristic results from that of CPLEX were found to be within acceptable ranges. The solutions provided by DM and HM were compared, and the DM was found to perform better than the HM in terms of solution time and quality of solution. Encouraged by these results, we devised a GRASP (Greedy Randomized Adaptive Search Procedure) based DM (GDM), and employed it in a multi-start framework for the overall algorithm. On the whole, GDM performs better than DM and HM in terms of solution quality, but with longer solution times.

As stated in chapter I, design of SONET rings would involve formation of disjoint rings and an interconnecting main ring that connects all the disjoint rings together.

This research concentrated only on the formation of disjoint rings. To further the scope of the problem, constraints to form the main connecting ring could be analyzed as the next step in this field of research. Also, for the problem considered in this research, the test data was created assuming 100 percent connectivity in the network. Study could be made on a graph with less than 100 percent connectivity. Finally, more extensive and rigorous experimentation with real time data could be performed to have a better understanding of the results for the real world problems.

REFERENCES

Aarts, E., Lenstra, J. K. (Eds.), 1997. Local Search in Combinatorial Optimization. John Wiley and Sons, New York, NY.

Ahuja, R., Ergun, O., Orlin, J. B., Punnen, A., 2002. A survey of very large-scale neighborhood search techniques. Disrete Applied Mathematics 123, 75-102.

Ahuja, R., Orlin, J. B., Pallottino, S., Scaparra, M., Scutella, M., 2004. A multiexchange heuristic for the single-source capacitated facility location problem. Management Science 50, 749-760.

Ahuja, R., Orlin, J. B., Sharma, D., 2000. Very large-scale neighborhood search. International Transactions in Operational Research 7, 301-317.

Ahuja, R., Orlin, J. B., Sharma, D., 2003. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. Operations Research Letters 31, 185-194.

Ahuja, R. K., Magnanti, T. L., Orlin, J. B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Upper Saddle River, NJ.

Bellmore, M., Hong, S., 1974. Transformation of Multisalesmen Problem to the Standard Traveling Salesman Problem. Journal of the Association for Computing Machinery 21, 500-504.

Branco, I. M., Coelho, J. D., 1990. The Hamiltonian p-median problem. European Journal of Operational Research 47, 86-95.

Christofides, N., 1976. Worst-case analysis for a new heuristic for the travelingsalesman problem. Tech. Rep. 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Feo, T., Resende, M. G. C., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109-133.

Gavish, B., Srikanth, K., 1986. Optimal solution method for large scale multiple traveling salesman problem. Operations Research 34 (5), 698-717.

Glaab, H., Pott, A., 2000. The Hamiltonian p-median problem. The Electronic Journal of Combinatorics 7, 1–25.

Gutin, G., Punnen, A. P. (Eds.), 2002. The Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Hansen, P., Hegedahl, B., Hjortkjaer, S., Obel, B., 1994. A heuristic solution to the warehouse location-routing problem. European Journal of Operational Research 76, 111-127.

Hong, S., Padberg, M. W., 1977. A note on the symmetric multiple traveling salesman problem with fixed charges. Operations Research 25 (5), 871-874.

Jacobsen, S., Madsen, O., 1980. A comparative study of heuristics for a two-level routing-location problem. European Journal of Operational Research 34, 378-387.

Jonker, R., Volgenat, T., 1986. An improved transformation of the symmetric multiple traveling salesman problem. Operations Research 34 (5), 698-717.

Laguna, M., 1994. Clustering for the design of SONET rings in interoffice telecommunications. Management Science 40 (11), 1533-1541.

Laporte, G., 1988. Location-routing problems. In: Golden, B., Assad, A. A. (Eds.), Vehicle Routing: Methods and Studies. North-Holland, Amsterdam, pp. 163-198.

Lin, C., Chow, C., Chen, A., 2002. A location routing loading problem for bill delivery services. Computers & Industrial Engineering 43, 5-25.

Lin, S., Kernighan, B. W., 1973. An effective heuristic algorithm for the traveling salesman problem. Operations Research 21, 498-516.

Luss, H., Rosenwein, M. B., Wong, R. T., 1998. Topological network design for SONET ring structure. IIE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 28 (6), 780-790.

Madsen, O., 1983. Methods for solving combined two level location-routing problems of realistic dimensions. European Journal of Operational Research 12, 295-301.

Michalewicz, Z., Fogel, D. B., 2000. How to Solve It: Modern Heuristics. Springer-Verlag, Berlin, Germany.

Min, H., Jayaraman, V., Srivastava, R., 1998. Combined location-routing problem: A synthesis and future research directions. European Journal of Operational Research 108, 1-15.

Perl, J., Daskin, M. S., 1983. A warehouse location problem. Transportation Research Quarterly, Part BMethodological 19, 381-396.

Resende, M. G. C., Riberio, C. C., 2003. Greedy randomized adaptive search procedures. In: Glover, F., Kochenberger, G. A. (Eds.), Handbook of Metaheuristics. Kluwer Academic Publishers, Norwell, MA, pp. 219–249.

Salhi, S., Rand, G., 1989. The effect of ignoring routes when locating depots. European Journal of Operational Research 39, 150-156.

Soriano, P., Wynants, C., Seguin, R., Labbe, M., Gendreau, M., Fortz, B., 1998. Design and dimensioning of survivable SDH/SONET networks. In: Sans'o, B., Soriano, P. (Eds.), Telecommunications Network Planning. Center for Research on Transportation, University of Montreal, Montreal, Canada, pp. 571-584.

Srivastava, R., 1993. Alternate solution procedures for the location-routing problem. International Journal of Management Science 21, 497-506.

Thompson, P., Orlin, J., 1989. The theory of cyclic transfers. Working paper - OR200– 89. Accessed by http://web.mit.edu/jorlin/www/oldpapersfolder/cyclic_tranfers.pdf.

Thompson, P., Psaraftis, H., 1993. Cyclic transfer algorithms for multi-vehicle routing and scheduling problems. Operations Research 41, 935-946.

Tuzun, D., Burke, L., 1999. A two-phase tabu search approach to the location routing problem. European Journal of Operational Research 116, 87-99.

Wu, T., Low, C., Bai, J., 2002. Heuristic solutions to multi-depot location-routing problems. Computers & Operations Research 29, 1393-1415.

APPENDIX A

INDIVIDUAL RESULTS FOR TEST PROBLEMS

Table V Percentage Gaps for Construction Heuristics - 15 and 18 Nodes

15 Node Problem							
c = 2		c = 3		c = 4		c = 5	
DM	HM	DM	HM	DM	HM	DM	HM
14.02	49.56	36.34	52.16	45.58	68.04	57.39	90.51
2.79	27.04	21.98	46.70	38.45	14.07	54.31	23.83
28.69	60.01	54.69	44.78	54.24	66.77	36.16	70.93
22.54	34.60	36.90	27.90	27.40	59.91	16.13	20.95
1.61	18.16	23.30	70.47	19.65	84.69	4.15	34.44
12.75	22.69	13.46	34.45	23.79	52.04	4.43	65.34
3.00	25.29	67.02	90.65	11.78	35.72	42.80	61.39
3.14	3.14	26.44	30.97	19.43	24.93	26.06	0.00
15.01	21.55	34.68	52.54	6.59	71.25	10.33	76.74
7.76	48.52	51.97	37.23	44.60	48.30	56.54	67.29
18 Node Problem							
c = 2		c = 3		c = 4		c = 5	
DM	HM	DM	HM	DM	HM	DM	HM
13.67	47.44	40.82	71.73	55.08	90.36	59.24	117.42
13.46	26.46	47.79	37.33	51.64	85.82	72.98	60.63
30.63	61.54	46.04	93.19	37.14	31.58	14.69	70.25
3.96	10.15	33.36	22.61	17.22	52.23	69.95	92.08
4.60	6.93	12.15	34.27	31.49	60.90	14.85	66.59
11.27	30.85	13.20	40.70	25.77	49.80	39.13	22.61
10.10	34.07	41.27	52.17	64.41	53.80	114.17	48.62
5.43	47.13	14.01	65.80	50.77	87.58	97.64	71.29
12.65	20.12	36.44	30.57	10.67	51.35	26.48	110.13
18.53	38.11	38.19	37.61	52.36	70.79	63.25	66.22

Table VI Percentage Gaps for Construction Heuristics - 21 and 24 Nodes

21 Node Problem							
c = 2		c = 3		c = 4		c = 5	
DM	HM	DM	HM	DM	HM	DM	HM
11.09	10.65	37.13	61.68	40.89	72.54	85.40	123.10
18.23	43.48	35.92	54.27	45.23	88.55	74.98	83.67
28.75	12.23	63.53	36.65	33.98	39.67	47.42	51.90
3.35	3.35	63.47	39.31	29.42	32.42	46.15	124.92
8.90	4.83	34.57	35.06	25.36	73.15	36.92	55.48
27.28	30.18	29.05	35.30	5.95	55.29	42.25	62.50
19.53	44.17	18.91	47.83	29.57	49.95	59.07	63.32
5.74	27.52	16.74	40.81	55.80	107.90	46.28	120.98
0.00	28.17	43.83	66.99	55.59	84.24	26.76	125.03
36.05	40.39	43.67	69.48	13.27	40.07	49.02	77.01
24 Node Problem							
c = 2		c = 3		c = 4		c = 5	
DM	HM	DM	HM	DM	HM	DM	HM
9.47	24.93	51.27	78.96	54.43	94.89	22.33	114.13
15.31	37.86	59.18	46.96	53.75	72.95	NA	NA
24.36	26.36	70.28	60.05	78.91	64.33	90.11	55.37
8.34	6.40	62.96	38.93	30.06	52.70	NA	NA
8.20	13.93	26.20	20.07	29.78	49.35	30.21	60.69
24.94	22.92	50.19	35.03	44.55	66.62	45.90	70.48
15.39	4.62	18.32	58.60	41.95	48.60	36.87	56.87
12.35	33.25	14.96	31.39	45.19	84.42	2.85	93.10
4.88	6.34	43.24	67.82	65.06	93.10	47.27	117.61
23.21	29.70	48.66	52.70	35.94	84.48	37.09	87.39

Table VII Percent Gaps and Run-times (Secs) for Improvement Heuristics (15 Node - 2 and 3 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 2									
20.56	3.44	0.00	1.91	0.00	7.89	0.00	3.34	0.00	8.00
21.72	2.83	2.42	1.59	2.42	3.73	0.39	3.45	0.39	3.84
43.18	3.03	4.44	3.60	4.44	4.25	0.95	3.69	0.00	4.31
11.26	3.74	11.26	3.77	9.13	4.78	0.00	2.39	6.18	4.92
12.04	4.22	1.61	2.89	1.61	4.66	0.00	3.78	0.00	4.78
9.17	2.30	5.98	1.97	5.98	3.66	5.98	2.47	5.98	3.75
3.00	1.47	3.00	1.72	3.00	2.92	0.00	2.36	0.00	3.06
3.14	0.98	3.14	1.16	3.14	4.17	0.00	1.56	0.00	4.27
11.06	1.45	12.70	4.23	5.30	6.25	11.30	5.28	5.30	6.31
5.09	5.50	7.22	1.83	5.09	6.38	6.84	1.95	0.00	6.70
C = 3									
1.89	6.41	9.36	4.09	1.89	13.98	8.05	4.27	0.00	14.23
3.80	5.41	2.44	4.13	2.44	7.78	0.00	4.06	0.00	7.89
4.50	2.23	4.50	9.41	4.50	10.31	0.00	8.89	0.00	10.42
2.46	3.09	3.28	6.12	0.98	4.63	1.74	5.49	0.00	4.94
2.65	12.14	2.65	4.20	2.65	10.05	2.65	4.13	2.65	10.17
19.49	2.56	0.00	4.17	0.00	9.64	0.00	3.75	0.00	9.77
10.88	4.31	4.05	5.44	4.05	7.41	0.00	5.14	0.00	7.53
4.37	5.56	4.37	5.50	4.37	8.89	0.00	5.45	0.00	8.98
4.28	7.41	0.28	7.11	4.28	13.47	4.28	6.91	4.28	13.73
1.31	3.36	1.31	6.50	1.31	10.19	0.00	6.34	0.00	10.55

Table VIII Percent Gaps and Run-times (Secs) for Improvement Heuristics (15 Node - 4 and 5 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 4									
21.01	8.11	4.06	11.13	4.06	24.09	0.40	10.83	0.40	24.23
14.07	0.77	0.00	4.02	0.00	14.09	0.00	4.33	0.00	14.20
0.09	5.78	0.09	7.23	0.09	29.33	0.00	7.50	0.00	29.47
0.57	6.92	12.24	1.70	0.57	6.27	12.21	1.94	0.00	6.39
1.67	36.55	1.67	3.97	1.67	10.63	0.00	4.02	0.00	10.86
0.00	13.84	3.12	3.86	0.00	15.05	2.82	3.94	0.00	15.17
11.03	2.02	1.34	1.80	1.34	6.03	0.00	2.05	0.00	6.19
1.60	2.77	1.60	3.89	1.60	7.30	0.00	3.81	0.00	7.55
0.47	36.98	3.36	3.22	0.47	9.87	2.18	3.23	0.00	10.16
8.77	5.33	0.00	4.99	0.00	36.87	0.00	4.70	0.00	37.01
C = 5									
0.00	379.96	0.00	85.75	0.00	82.58	0.00	81.25	0.00	82.69
0.00	1.25	0.00	8.66	0.00	62.80	0.00	7.92	0.00	62.91
0.00	52.91	0.00	4.08	0.00	10.69	0.00	4.13	0.00	10.77
0.00	2.67	0.00	2.15	0.00	5.92	0.00	2.21	0.00	6.05
0.00	13.28	0.00	1.63	0.00	20.56	0.00	1.77	0.00	20.81
0.00	158.56	0.00	1.63	0.00	2.56	0.00	1.74	0.00	2.66
0.00	7.05	0.00	2.45	0.00	7.86	0.00	2.92	0.00	7.95
0.00	0.52	0.00	4.45	0.00	35.86	0.00	4.59	0.00	35.95
0.00	209.51	0.00	1.31	0.00	11.17	0.00	1.56	0.00	11.25
0.00	84.01	0.00	41.15	0.00	103.15	0.00	51.09	0.00	103.26

Table IX Percent Gaps and Run-times (Secs) for Improvement Heuristics (18 Node-2 and 3 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 2									
10.54	10.31	4.90	1.61	4.90	12.83	0.00	2.00	0.00	12.94
10.47	10.11	11.06	1.23	11.06	7.36	8.48	1.55	8.48	7.50
41.59	3.83	1.49	5.86	1.49	20.37	0.00	6.23	0.00	20.48
3.96	4.66	3.96	3.08	2.69	12.94	0.88	3.94	0.00	13.12
1.68	8.25	0.54	4.64	0.54	18.14	0.00	5.42	0.00	18.20
10.87	5.50	4.97	5.38	0.81	11.75	3.25	5.73	0.00	12.00
4.14	4.77	4.14	5.58	4.14	12.86	3.42	5.55	3.42	13.14
8.99	5.78	5.43	9.14	5.43	16.34	0.00	8.05	0.00	16.62
6.25	6.33	6.25	8.39	6.25	13.58	0.00	7.47	0.00	13.72
4.47	12.44	4.47	5.20	4.47	21.44	4.47	4.48	4.47	21.53
C = 3									
4.29	22.38	16.14	9.77	4.29	36.14	15.33	8.83	0.00	36.26
21.78	7.56	6.24	13.81	6.24	22.98	3.56	12.59	3.56	23.38
10.88	18.19	12.37	7.52	12.37	23.06	10.58	7.52	10.58	23.20
5.99	2.50	1.57	5.20	1.57	14.30	0.10	4.64	0.10	14.45
1.80	4.86	0.83	6.16	0.83	33.69	0.83	5.58	0.83	33.84
1.31	16.66	1.31	5.95	1.31	36.12	0.51	5.64	0.51	36.28
26.56	12.36	13.49	7.33	13.49	48.66	10.15	7.27	10.15	49.17
12.34	12.41	6.56	11.12	6.56	45.59	0.71	12.14	0.71	45.81
10.69	5.91	4.14	6.31	2.00	37.92	2.57	7.58	2.00	38.12
2.31	7.98	2.31	10.30	2.31	48.00	2.31	10.78	2.31	48.19

Table X Percent Gaps and Run-times (Secs) for Improvement Heuristics (18 Node-4 and 5 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 4									
2.28	83.90	9.29	17.55	9.29	192.81	5.85	17.08	5.85	192.96
15.12	36.39	5.08	8.78	5.08	57.89	0.00	8.72	0.00	58.03
2.69	7.05	2.12	9.61	2.12	31.08	1.29	9.17	1.29	31.23
0.55	6.94	0.55	2.08	0.55	23.47	0.00	2.50	0.00	23.61
8.99	45.80	4.87	12.63	0.38	34.05	4.87	13.06	0.38	34.34
12.14	10.72	1.58	6.00	1.58	94.23	0.93	6.92	0.93	94.33
4.22	14.36	7.26	28.96	6.60	33.06	6.90	30.98	1.69	33.20
0.00	41.53	19.05	11.68	0.00	27.03	18.48	11.91	0.00	27.22
10.10	26.36	7.97	2.36	0.00	32.86	6.79	2.42	0.00	33.06
1.15	46.22	1.95	10.75	0.55	175.17	1.95	14.00	0.00	175.34
C = 5									
0.05	1476.93	12.96	78.66	0.42	243.09	12.90	89.97	0.42	243.35
4.79	38.37	4.68	73.17	4.68	67.22	3.18	80.72	3.18	67.64
0.08	160.90	0.08	6.72	0.08	34.03	0.00	7.53	0.00	34.25
0.61	266.37	0.61	68.40	0.61	98.12	0.00	70.62	0.00	98.53
0.16	171.14	0.79	3.89	0.16	90.97	0.00	5.86	0.16	91.19
4.08	4.89	1.91	13.27	0.83	88.31	1.91	19.66	0.00	88.51
14.31	6.42	4.87	307.28	4.87	466.30	2.09	329.16	2.09	466.49
5.11	40.155	0.00	87.92	0.00	74.80	0.00	94.16	0.00	74.97
10.04	2127.24	1.18	10.94	1.18	138.48	0.00	10.45	0.00	138.84
0.00	125.81	0.00	31.86	0.00	68.36	0.00	29.14	0.00	68.56

Table XI Percent Gaps and Run-times (Secs) for Improvement Heuristics (21 Node - 2 and 3 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 2									
7.54	4.66	4.19	2.69	4.19	18.91	0.00	3.44	0.00	19.16
9.21	20.94	15.89	4.84	10.62	29.75	10.05	5.81	0.69	30.00
4.08	4.69	7.26	12.75	6.05	32.39	1.11	14.52	6.05	32.59
3.35	3.25	3.35	3.203	3.35	28.09	0.00	3.96	0.00	28.36
0.59	5.84	2.57	5.03	0.87	38.16	1.88	4.28	0.00	38.51
10.69	5.72	0.55	14.16	2.07	57.73	0.42	13.61	1.87	57.91
4.77	12.78	4.77	8.25	4.77	31.23	0.00	7.95	0.00	31.58
5.74	10.39	5.74	14.69	5.74	57.17	0.00	13.87	0.00	57.47
9.91	4.19	0.00	12.14	13.25	52.26	0.00	12.44	12.72	52.41
14.32	10.59	18.20	6.5	9.80	40.44	14.32	7.26	2.51	40.81
C = 3									
4.53	20.11	16.34	18.72	4.53	104.31	10.96	18.28	0.00	104.72
7.64	21.75	9.01	15.77	7.64	81.62	1.99	15.03	0.00	81.86
24.82	9.84	4.18	39.92	4.18	68.45	0.36	38.52	0.36	68.75
2.44	6.00	20.40	21.34	20.40	43.80	20.30	22.37	20.30	44.45
7.02	12.91	2.80	21.61	2.80	40.41	2.49	21.38	2.49	40.53
6.42	23.00	3.93	13.80	3.93	65.00	3.16	13.97	3.16	65.11
23.57	26.53	8.16	9.11	8.16	74.26	4.93	8.63	4.93	74.48
2.52	22.73	8.25	18.15	4.45	70.28	1.40	18.17	3.05	70.48
2.43	61.44	2.43	27.30	2.43	112.26	2.16	31.50	2.16	112.43
29.11	22.36	20.66	10.31	4.58	69.20	20.66	12.28	3.41	69.37

Table XII Percent Gaps and Run-times (Secs) for Improvement Heuristics (21 Node - 4 and 5 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 4									
1.94	115.31	9.08	14.19	1.94	488.68	7.64	15.34	0.00	488.88
42.57	40.81	4.38	12.99	4.38	405.97	3.46	13.17	3.46	406.13
12.86	16.55	6.34	14.50	4.72	122.98	4.59	14.48	1.89	123.15
29.71	3.97	1.43	10.11	1.43	45.67	0.00	8.84	0.00	45.84
2.16	261.09	7.47	13.71	6.40	140.86	7.39	12.03	4.95	141.17
14.34	39.56	2.64	7.61	2.64	22.58	0.00	6.41	0.00	22.75
7.33	31.98	5.09	13.41	6.35	143.15	1.37	11.41	0.00	143.28
0.53	297.65	0.53	24.93	0.53	101.09	0.00	25.55	0.00	101.20
5.55	298.18	2.07	110.38	2.06	545.06	0.01	113.47	2.06	545.22
7.23	18.73	9.46	8.16	1.63	240.24	9.46	8.75	1.63	240.40
C = 5									
12.78	1664.54	28.76	152.06	15.44	2292.82	28.14	167.48	14.80	2293.11
9.07	338.05	1.71	191.84	1.71	344.35	0.00	204.15	0.00	344.62
4.61	86.65	22.86	31.99	2.50	286.73	21.12	38.78	1.25	287.10
28.57	1611.49	6.27	7.39	6.27	246.21	6.27	9.67	6.27	246.37
1.31	97.42	12.97	29.31	1.31	302.09	12.79	33.00	0.00	302.38
0.00	115.20	0.00	32.34	0.00	529.85	0.00	32.20	0.00	529.97
11.68	64.45	4.12	25.47	4.92	1252.08	2.09	27.70	0.63	1252.23
3.20	4758.97	6.31	58.46	2.34	206.66	5.70	61.21	2.05	206.77
14.71	4737.15	1.16	11.19	1.16	115.69	0.00	11.31	0.00	115.86
8.45	651.03	0.40	48.09	0.40	304.06	0.00	40.92	0.00	304.20

Table XIII Percent Gaps and Run-times (Secs) for Improvement Heuristics (24 Node - 2 and 3 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 2									
10.17	13.92	10.07	21.63	6.19	25.34	4.29	19.59	0.00	25.55
19.92	36.94	10.54	11.30	10.54	36.56	4.57	10.67	4.57	37.05
1.09	14.44	9.16	15.50	1.09	20.95	1.04	14.75	0.00	21.09
6.40	4.80	6.40	11.67	6.40	40.45	0.00	12.25	0.00	40.59
4.20	6.53	4.20	8.02	4.20	45.67	3.57	8.77	3.57	45.80
1.01	10.73	17.75	17.12	1.01	52.53	12.41	16.69	0.86	52.87
0.22	6.45	4.14	7.14	4.14	38.11	3.06	7.44	3.06	38.25
7.05	6.89	8.73	14.95	6.35	44.05	4.51	17.91	1.09	44.17
13.62	12.27	11.86	30.64	1.91	46.22	9.08	34.73	1.19	46.55
18.31	25.29	12.11	13.83	3.71	38.84	3.69	14.26	0.35	38.95
C = 3									
16.53	3410.69	6.36	68.998	5.90	194.57	3.32	69.35	5.19	194.98
26.29	15.75	6.42	36.71	4.23	130.54	3.28	47.69	0.00	130.90
0.99	62.00	3.87	93.63	10.75	130.12	0.28	102.45	7.26	130.40
13.90	22.64	25.32	33.85	25.32	142.25	21.83	44.06	21.83	142.65
16.17	17.05	8.37	15.38	3.31	143.43	5.51	20.22	3.31	143.86
12.99	11.45	4.65	27.37	4.65	171.34	0.00	38.17	0.00	171.61
7.15	46.70	7.15	12.99	7.15	64.50	3.20	15.97	3.20	64.69
20.53	10.09	8.17	11.44	4.39	284.24	7.13	14.37	3.09	284.54
14.27	53.02	2.80	30.57	2.80	208.95	2.42	36.53	2.42	209.18
18.77	41.12	5.29	29.74	4.55	236.23	0.00	38.36	1.94	236.59

Table XIV Percent Gaps and Run-times (Secs) for Improvement Heuristics (24 Node - 4 and 5 Rings)

HM-Imp		DM-Imp		GDM-Imp		DMOpt-Imp		GDMOpt-Imp	
C = 4									
8.66	204.56	4.72	61.08	4.72	150.84	2.53	75.73	2.53	151.23
5.85	189.12	1.95	74.38	1.95	587.08	1.95	91.73	1.95	587.42
20.68	115.67	21.25	274.70	1.86	857.32	19.04	283.23	1.16	857.73
4.17	22.86	4.17	19.72	4.17	161.68	1.35	18.47	1.35	161.98
5.64	87.44	4.38	33.16	4.38	165.28	4.09	31.05	4.09	165.43
7.59	180.70	0.15	48.17	0.15	452.61	0.00	54.00	0.00	452.85
9.68	53.12	7.10	33.78	7.10	195.35	1.13	42.66	1.13	195.56
12.87	448.02	4.07	49.77	8.89	256.71	0.00	64.19	3.87	256.90
30.00	710.19	8.13	148.50	8.13	973.55	7.14	173.81	7.14	973.71
3.39	456.10	10.30	22.47	3.39	389.24	8.45	23.28	0.00	389.60
C = 5									
22.73	2917.10	2.19	7.75	2.19	345.62	0.00	8.25	0.00	345.87
NA	3177.84	NA	4782.77	NA	1900.95	NA	4803.77	NA	1901.26
8.14	323.70	3.31	3087.05	3.31	3063.45	0.00	1643.02	0.00	3063.63
NA	5451.06	NA	5269.34	NA	233.90	NA	5378.94	NA	234.07
10.16	1379.45	9.92	73.25	2.42	418.29	9.41	41.56	0.00	418.69
4.60	1214.59	0.00	115.94	0.00	284.46	0.00	67.17	0.00	284.77
8.29	134.29	5.47	53.84	5.47	556.64	3.17	34.16	3.17	556.86
2.85	5348.63	2.85	3.48	2.85	158.87	0.00	2.45	0.00	159.03
4.61	10780.90	3.51	522.13	3.51	1420.10	2.43	394.43	2.43	1420.28
17.66	1645.16	1.33	62.26	1.33	1695.39	0.00	65.41	0.00	1695.58

VITA

Sarath K. Sasi Kumar, was born in Alwaye, Kerala, India. He graduated from Regional Engineering College, Trichy, India, in June 2002 with a Bachelor of Engineering in instrumentation & control engineering. He entered Texas A&M University in the Master of Science program in industrial engineering in Fall 2003, and will be graduating in August 2005.

His permanent address is 120, Thiruvengada Nagar, Trichy, Tamil Nadu, 620013, India.

The typist for this thesis was Sarath K Sasi Kumar.